Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

«Інститут прикладного системного аналізу»

Кафедра математичних методів системного аналізу

Звіт

про виконання лабораторної роботи №2

з дисципліни

«Розпізнавання образів»

Виконали: студенти IV курсу

групи КА-76

Панасюк Я.І.

та групи КА-76

Іванов С.

Перевірила:

Дідковська М.В.

Київ – 2020

Завдання

Кожен учасник або учасниця команди спершу обирає дескриптор (один із розглянутих у лекціях або ж знайдений окремо) та предмет на прикладі якого відбуватиметься дослідження. Враховуючи що вас багато, будь ласка обирайте унікальніші предмети за улюблену чашку/телефон/мишку. Маючи те і інше напоготові кожен учасник бригади має зняти не менше сотні фото предмета, варіюючи його розміщення та ракурс в кадрі, освітлення, наявність візуальних перешкод, зашакаленість зображення, фокусну віддаль та тремтіння рук. Сотня фото обраного предмету на однаковій сцені з однаковою якістю зйомки, але з різних ракурсів на жаль не підійде, постарайтесь наполегливо варіювати сцени і умови зйомки. До цих фото варто додати невелику підбірку зображень, що не містять предмет, або ж містять предмет візуально подібний до вашого, штук 20 повинно вистачити, якщо залишиться натхнення можна й більше. Після чого ми нарешті дійшли до цікавого, а саме до дослідження: Вам потрібно згенерувати обраний дескриптор для обраного предмета, після чого з його допомогою розпізнати об’єкт на всій тестовій вибірці збираючи при цьому такі метрики: відносна кількість правильно суміщених ознак, похибка локалізації (відстань між реальним розміщенням предмета в кадрі та розпізнаним) та відносний час обробки фото в залежності від розміру зображення. Метрики мають зберегтись у файлику для подальших досліджень. Наступним кроком ви обмінюєтесь об’єктом з колегою, і уже маючи готову збиралку метрик, обчислюєте їх для предмета вашого сусіда, таким чином у вас збирається 9 наборів даних, по три на дескриптор. Самою ж ідеєю лаби є дослідити розбіжності у роботі ваших дескрипторів та виконати порівняльний аналіз їх поведінки, сформулювати висновки з викладками і прикладами так аби було зрозуміло вам та, сподіваюсь, усім вашим колегам. Таким чином кінцевим результатом буде від вас гуглдок з описом виняткових особливостей, сильних та слабких сторін дескриптора і обгрунтуванням чому вони поводяться саме так.

Хід роботи

Вибірки об'єктів

In [1]:
```python
import glob
import cv2

#DUCK_DS
duck_train_pth = glob.glob("duck_ds/train/*.jpg")
duck_test_pth = glob.glob("duck_ds/test/*.jpg")
duck_ds_train = [cv2.imread(_) for _ in duck_train_pth]
duck_ds_test = [cv2.imread(_) for _ in duck_test_pth]
reduced_duck_ds_train = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in duck_train_pth]
reduced_duck_ds_test = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in duck_test_pth]

#CUP_DS
cup_train_pth = glob.glob("cup_ds/train/*.jpg")
cup_test_pth = glob.glob("cup_ds/test/*.jpg")
cup_ds_train = [cv2.imread(_) for _ in cup_train_pth]
cup_ds_test = [cv2.imread(_) for _ in cup_test_pth]
reduced_cup_ds_train = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in cup_train_pth]
reduced_cup_ds_test = [cv2.resize(cv2.imread(_), (0,0), fx=0.5, fy=0.5) for _ in cup_test_pth]
```

Дескриптори

ORB (Oriented FAST and Rotated BRIEF)

In [2]:
```python
import numpy as np
import cv2
import time

def ORB_descript(query_img, train_img, show_final_img=False):
    # Read the query image as query_img and train image as train_img
    # This query image is what you need to find in train image
    # Convert it to grayscale
    query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
    train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)

    # Initialize the ORB detector algorithm
    orb = cv2.ORB_create()

    t_start = time.time()

    # Now detect the keypoints and compute
    # the descriptors for the query image and train image
    queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw,None)
    trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)

    if(queryDescriptors is None or trainDescriptors is None):
        return 0, 966, 0

    # Initialize the Matcher for matching
    # the keypoints and then match the keypoints
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = matcher.match(queryDescriptors,trainDescriptors)

    t_end = time.time()
    # Metrics
    # the relative processing time of the image depending on the image size
    rel_time = t_end - t_start

    # Sort them in the order of their distance
    matches = sorted(matches, key = lambda x:x.distance)

    if(show_final_img):
        # draw the matches to the final image
        # containing both the images the drawMatches()
        # function takes both images and keypoints
        # and outputs the matched query image with
        # its train image
        final_img = cv2.drawMatches(query_img, queryKeypoints,
        train_img, trainKeypoints, matches[:20],None)

        final_img = cv2.resize(final_img, (1000,650))

        # Show the final image
        cv2.imshow("Matches", final_img)
        cv2.waitKey(3000)

    # Metrics
    # the relative number of correctly matched features
    rel_numb = len(matches)*2 / (len(queryDescriptors) + len(trainDescriptors))

    # localization inaccuracy
    distances = [_.distance for _ in matches]
    avr_dist = sum(distances)/len(distances)

    return rel_numb, avr_dist, rel_time
```

BRIEF з використанням детектору CenSurE (або STAR в opencv)

In [3]:
```python
import numpy as np
import cv2
import time


def BRIEF_descript(query_img, train_img, show_final_img=False):
    # Read the query image as query_img and train image as train_img
    # This query image is what you need to find in train image
    # Convert it to grayscale
    query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
    train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)

    # Initiate STAR detector
    star = cv2.xfeatures2d.StarDetector_create()
    # Initiate BRIEF extractor
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

    t_start = time.time()

    # find the keypoints with STAR
    queryKeypoints = star.detect(query_img_bw,None)
    trainKeypoints = star.detect(train_img_bw,None)
    # compute the descriptors with BRIEF
    queryKeypoints, queryDescriptors = brief.compute(query_img_bw, queryKeypoints)
    trainKeypoints, trainDescriptors = brief.compute(train_img_bw,  trainKeypoints)

    if(queryDescriptors is None or trainDescriptors is None):
        return 0, 966, 0

    # Initialize the Matcher for matching
    # the keypoints and then match the keypoints
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = matcher.match(queryDescriptors,trainDescriptors)

    t_end = time.time()
    # Metrics
    # the relative processing time of the image depending on the image size
    rel_time = t_end - t_start

    # Sort them in the order of their distance
    matches = sorted(matches, key = lambda x:x.distance)

    if(show_final_img):
        # draw the matches to the final image
        # containing both the images the drawMatches()
        # function takes both images and keypoints
        # and outputs the matched query image with
        # its train image
        final_img = cv2.drawMatches(query_img, queryKeypoints,
        train_img, trainKeypoints, matches[:20],None)

        final_img = cv2.resize(final_img, (1000,650))

        # Show the final image
        cv2.imshow("Matches", final_img)
        cv2.waitKey(3000)

    # Metrics
    # the relative number of correctly matched features
    rel_numb = len(matches)*2 / (len(queryDescriptors) + len(trainDescriptors))

    # localization inaccuracy
    distances = [_.distance for _ in matches]
    avr_dist = sum(distances)/len(distances)

    return rel_numb, avr_dist, rel_time
```

Метрики

In [4]:
```python
import matplotlib.pyplot as plt
from bokeh.plotting import figure, output_file, show
from pathlib import Path

def get_metrics(descriptor, ds_train, ds_test, ds_name, descr_name, plotting_graphs=True):
    """It creates files with "the relative number of correctly matched features",
    "localization inaccuracy" and "relative processing time" metrics"""
    rel_numb_of_correct_mathced_feat=[]
    distances=[]
    relative_time=[]

    for train_img in ds_train:
        distances_to_train_img =[]
        rel_numb_for_train_img =[]
        time_for_train_img =[]

        for test_img in ds_test:
            rel_numb, avr_dist, time = descriptor(train_img, test_img)

            rel_numb_for_train_img.append(rel_numb)
            distances_to_train_img.append(avr_dist)
            time_for_train_img.append(time)

        rel_numb_of_correct_mathced_feat.append(rel_numb_for_train_img)
        distances.append(distances_to_train_img)
        relative_time.append(time_for_train_img)

    # Create directories for output data
    Path(f"{ds_name}/metrics/{descr_name}/data").mkdir(parents=True, exist_ok=True)
    Path(f"{ds_name}/metrics/{descr_name}/graphs").mkdir(parents=True, exist_ok=True)
    print("Data and graphs")
    # write metrics for every train img
    for i in range(len(ds_train)):
        with open(f"{ds_name}/metrics/{descr_name}/data/train_img{i}_rel_numb.txt", "w") as file1:
            file1.write("\n".join([str(_) for _ in rel_numb_of_correct_mathced_feat[i]]))
        with open(f"{ds_name}/metrics/{descr_name}/data/train_img{i}_avr_dist.txt", "w") as file2:
            file2.write("\n".join([str(_) for _ in distances[i]]))
        with open(f"{ds_name}/metrics/{descr_name}/data/train_img{i}_relative_time.txt", "w") as file3:

            file3.write("\n".join([str(_) for _ in relative_time[i]]))

    if(plotting_graphs):
        arr=[_ for _ in range(len(ds_test))]

        for i in range(len(ds_train)):
            output_file(f"{ds_name}/metrics/{descr_name}/graphs/train_img{i}_rel_numb_graph.html")
            p = figure(title=f"the relative number of correctly matched features from train_img_{i}" ,
x_axis_label='test_img_id', y_axis_label='rel numb')
            p.line(arr, rel_numb_of_correct_mathced_feat[i], legend_label="Temp.", line_width=2)
            show(p)
        for i in range(len(ds_train)):
            output_file(f"{ds_name}/metrics/{descr_name}/graphs/train_img{i}_distance_graph.html")
            p = figure(title=f"Distance to train_img_{i}" , x_axis_label='test_img_id', y_axis_label='d
istance')
            p.line(arr, distances[i], legend_label="Temp.", line_width=2)
            show(p)
        for i in range(len(ds_train)):
            output_file(f"{ds_name}/metrics/{descr_name}/graphs/train_img{i}_relative_time_graph.html")
            p = figure(title=f"Relative time for train_img_{i}" , x_axis_label='test_img_id', y_axis_la
bel='relative_time')
            p.line(arr, relative_time[i], legend_label="Temp.", line_width=2)
            show(p)
```

Отримання метрик

In [10]:
```python
#duck_ds + ORB descriptor
get_metrics(ORB_descript, duck_ds_train, duck_ds_test, "duck_ds", "ORB")
```

In [6]:
```python
#cup_ds + ORB descriptor
get_metrics(ORB_descript, cup_ds_train, cup_ds_test, "cup_ds", "ORB")
```

In [7]:
```python
#cup_ds + BRIEF descriptor
get_metrics(BRIEF_descript, cup_ds_train, cup_ds_test, "cup_ds", "BRIEF")
```

In [9]:
```python
#duck_ds + BRIEF descriptor
get_metrics(BRIEF_descript, duck_ds_train, duck_ds_test, "duck_ds", "BRIEF")
```

In [ ]:
```python
#resized duck_ds + ORB descriptor
get_metrics(ORB_descript, reduced_duck_ds_train, reduced_duck_ds_test, "duck_ds", "reduced_ORB")
```

In [ ]:
```python
#resized cup_ds + ORB descriptor
get_metrics(ORB_descript, reduced_cup_ds_train, reduced_cup_ds_test, "cup_ds", "reduced_ORB")
```

In [ ]:
```python
#resized cup_ds + BRIEF descriptor
get_metrics(BRIEF_descript, reduced_cup_ds_train, reduced_cup_ds_test, "cup_ds", "reduced_BRIEF")
```

In [ ]:
```python
#resized duck_ds + BRIEF descriptor
get_metrics(BRIEF_descript, reduced_duck_ds_train, reduced_duck_ds_test, "duck_ds", "reduced_BRIEF")
```

Результати

Отримані метрики можна побачити у файлах:
[Назва датасету]_[Метрики]_[Назва дескриптору, наявність стиснення зображень]_[data]_[Номер тренованого зображення в масиві]
Граффіки на основі отриманих даних:
[Назва датасету]_[Метрики]_[Назва дескриптору, наявність стиснення зображень]_[graphs]_[Номер тренованого зображення в масиві]

Щоб вивести дескпиптор конкретного зображення, введіть дескриптор, назви датасетів та номер тренованого й тестового зображення у масиві.
Наприклад:

In [ ]:
```python
rel_numb, avr_dist  = ORB_descript(duck_ds_train[1], duck_ds_test[40], show_final_img=True)
```

Аналіз

З отриманих даних можемо зробити такі висновки:
- Похибка локалізації у ORB виявилась стабільно вищою, за відповідну похибку у BRIEF.
- Натомість, він здебільшого показав кращі показники правильності суміщених ознак.
- ORB також був швидшим за конкурента, хоч варто зауважити, що обидва алгоритми пошуку ознак можна вважати дуже швидкими, адже це бінарні алгоритми, що вирізняються значною перевагою у розрахунковій оптимізації в порівнянні з такими алгоритмами як SIFT та SURF, натомість програючи останнім в точності. Різниця в швидкості тут пов'язана із тим, що SIFT та SURF використовують градієнти областей навколо особливої точки, в той час як бінарні алгоритми описують цю облась двійковим рядком попарного порівняння яскравості пікселів, що прискорює обчислення.
- Стиснення зображень додатково значно пришвидшило час роботи обох алгоритмів, майже не вплинувши на інші показники.
- Обидва алгоритми виявились інваріантними до таких спотворень як зміна кута огляду, розмиття зображення, натомість досить значну зміну викликав поворот шуканого об'єкта.

ORB та BRIEF є прикладами бінарних алгоритмів пошуку особливих точок, тож, як було зазначено вище, вирізняються швидкістю роботи, оптимізацією розрахунків, програючи натомість в якості виявлення ознак. Ідейно, BRIEF має такі проблеми як неоптимальний вибір точок для розрахунку дескриптора і неможливість врахування орієнтації точки при розпізнаванні. Метод ORB покликаний усунути зазначені недоліки і бути покращеним аналогом методу BRIEF. Так, для розрахунку кута використовуються кооридинати центру ваги, а сама орієнтація подається вектором із початком в центральній точці і кінцем в центрі ваги, що розраховується через моменти зображення. Перевагою методу BRIEF є те, що дескриптори мають значну дисперсію та середнє значення близько 0,5, що допомагає із однотонними областями зображень, що зазвичай погано розпізнаються і легко корелюють з іншими. Щоб зберегти цю якість та некорельованість бінарних тестів, яку орієнтовані дескриптори втрачають, ORB використовує пошук серед усіх можливих бінарних тестів, щоб знайти ті, що мають високу дисперсію, середні значення близькі до 0,5 та є некорельованими.

Висновки

В ході виконання лабораторної роботи ми створили два датасети й побудували ORB та BRIEF дескриптори. Співставивши дескриптори train та test вибірок, ми змогли отримати метрики якості та швидкості роботи цих десприпторів, а також порівняти їх між собою.